Python

Overview

CellBase implements a Python client library called **PyCellBase** to query data through REST web services API. PyCellBase provides programmatic access to **all implemented** REST web services, providing an easy, lightweight, fast and intuitive access to all CellBase data in a user-friendly way without the need of installing any local database, you just need to configure the remote CellBase REST URL. Data is always available by a high-availability cluster and queries have been tuned to ensure a real-time performance. PyCellBase offers the convenience of an object-oriented scripting language and provides the ability to integrate the obtained results into other Python applications. More info about this package in the Python client tutorial section. PyCellBase uses *multithreading* to improve performance when the number of queries exceed a specific limit.

PyCellBase implements a *CellBaseClient* class factory can create all the **different clients** to the data we want to query (e.g. gene, transcript, variation, protein, genomic region, variant). Each of these clients implement different functions to query all REST web services. Most of these methods will need to be provided with **comma-separated IDs or list of IDs**. Optional filters and extra options can be added as key-value parameters. **Responses** are retrieved as **JSON formatted data**. Therefore, fields can be queried by key. **Configuration** data as host, API version, or species is stored in a ConfigClient object. A custom configuration can be passed to CellBaseClient with a ConfigClient object provided with a JSON or YAML config file. If you want to change the configuration on the fly you can directly modify the ConfigClient object.

PyCellBase is open-source and code can be found at `https://github.com/opencb/cellbase/tree/develop/clients/python/pycellbase`. PyCellbase be easily installed using PyPI. Please, find more details on how to use the python library at: Python client tutorial

Table of Contents:

Useful Links

- PyPI CellBase
- PyCellBase examples

Installation

Python client requires **Python 3.x,** although most of the code is fully compatible with Python 2.7. You can install PyCellBase either from PyPI repository or from the source code.

PyPI

PyCellBase client is deployed at PyPI and available at https://pypi.org/project/pycellbase. It can be easily installed using *pip* by executing:

```
$ pip install pycellbase
```

Source Code

PyCellBase can be installed from source code. You can get CellBase source code by cloning GitHub CellBase repository and executing *setup.py*:

```
## creates a folder called cellbase with the source code from branch
'master'
$ git clone -b master https://github.com/opencb/cellbase.git

## move to pycellbase folder and install
$ cd cellbase/clients/python
$ python setup.py install
```

Getting Started

Configuration

Configuration stores the REST services host, API version and species.

Getting the default configuration:

```
>>> ConfigClient().get_default_configuration()
{'version': 'v4', 'species': 'hsapiens', 'rest': {'hosts':
['http://bioinfo.hpc.cam.ac.uk:80/cellbase']}}
```

Showing the configuration parameters being used at the moment:

```
>>> cbc.show_configuration()
{'host': 'bioinfo.hpc.cam.ac.uk:80/cellbase', 'version': 'v4', 'species':
'hsapiens'}
```

A custom configuration can be passed to CellBaseClient with a ConfigClient object. JSON and YML files are supported:

```
>>> from pycellbase.cbconfig import ConfigClient
>>> from pycellbase.cbclient import CellBaseClient

>>> cc = ConfigClient('config.json')
>>> cbc = CellBaseClient(cc)
```

A custom configuration can also be passed as a dictionary:

```
>>> from pycellbase.cbconfig import ConfigClient
>>> from pycellbase.cbclient import CellBaseClient

>>> custom_config = {'rest': {'hosts': ['bioinfo.hpc.cam.ac.uk:80
/cellbase']}, 'version': 'v4', 'species': 'hsapiens'}
>>> cc = ConfigClient(custom_config)
>>> cbc = CellBaseClient(cc)
```

If you want to change the configuration on the fly you can directly modify the ConfigClient object:

```
>>> cc = ConfigClient()
>>> cbc = CellBaseClient(cc)

>>> cbc.get_config()['version']
'v4'

>>> cc.version = 'v3'
>>> cbc.get_config()['version']
'v3'
```

Querying data

The first step is to import the module and initialize the CellBaseClient:

```
## import pycellbase dependencies
from pycellbase.cbconfig import ConfigClient
from pycellbase.cbclient import CellBaseClient

## CellBaseClient object is a factory that allows to create all the other
clients
cbc = CellBaseClient()
```

CellBaseClient factory object allows to create all the other clients. The next step is to create a specific client to query CellBase, for instance to call to *Gene* web services:

```
gc = cbc.get_gene_client()
```

And now, you can start asking to the CellBase RESTful service by providing a query ID:

```
tfbs_responses = gc.get_tfbs('BRCA1')  # Obtaining TFBSs for BRCA1 gene
```

Responses are retrieved as JSON formatted data. Therefore, fields can be queried by key:

```
>>> tfbs_responses = gc.get_tfbs('BRCA1')
>>> tfbs_responses[0]['result'][0]['tfName']
'E2F4'

>>> transcript_responses = gc.get_transcript('BRCA1')
>>> 'Number of transcripts: %d' % (len(transcript_responses[0]['result']))
'Number of transcripts: 27'

>>> for tfbs_response in gc.get_tfbs('BRCA1,BRCA2,LDLR'):
...     print('Number of TFBS for "%s": %d' % (tfbs_response['id'], len
(tfbs_response['result'])))
'Number of TFBS for "BRCA1": 175'
'Number of TFBS for "BRCA2": 43'
'Number of TFBS for "LDLR": 141'
```

Data can be accessed specifying comma-separated IDs or a list of IDs:

```
>>> tfbs_responses = gc.get_tfbs('BRCA1')
>>> len(tfbs_responses)
1

>>> tfbs_responses = gc.get_tfbs('BRCA1,BRCA2')
>>> len(tfbs_responses)
2

>>> tfbs_responses = gc.get_tfbs(['BRCA1', 'BRCA2'])
>>> len(tfbs_responses)
2
```

If there is an available resource, but there is not an available method in this python package, the CellBaseClient can be used to create the URL of interest and query the RESTful service:

```
>>> tfbs_responses = cbc.get(category='feature', subcategory='gene',
query_id='BRCA1', resource='tfbs')
>>> tfbs_responses[0]['result'][0]['tfName']
'E2F4'
```

Optional filters and extra options can be added as key-value parameters (value can be a comma-separated string or a list):

```
>>> tfbs_responses = gc.get_tfbs('BRCA1')
>>> len(res[0]['result'])
175

>>> tfbs_responses = gc.get_tfbs('BRCA1', include='name,id')
>>> len(res[0]['result'])
175

>>> tfbs_responses = gc.get_tfbs('BRCA1', include = ['name', 'id'])
>>> len(res[0]['result'])
175

>>> tfbs_responses = gc.get_tfbs('BRCA1', limit=100)
>>> len(res[0]['result'])
100

>>> tfbs_responses = gc.get_tfbs('BRCA1', skip=100)
>>> len(res[0]['result'])
75
```

If there is an available resource, but there is not an available method in this python package, the **CellBas eClient** class can be used to **create the URL of interest**. This class is able to access the RESTful Web Services through the *get* method it implements. In this case, this method needs to be provided with those parameters which are required by the URL: category (e.g. feature), subcategory (e.g. gene), ID to search for (e.g. BRCA1) and method to query (e.g. search).

Integrated Help

The best way to know which data can be retrieved for each client is either checking out the RESTful web services section of the CellBase Wiki or the CellBase web services

If we do not know which method is the most adequate for our task, we can get helpful information for each data-specific client:

```
>>> cbc.get_region_client().get_help()
RegionClient
    - get_clinical: Retrieves all the clinical variants
    - get_conservation: Retrieves all the conservation scores
    - get_gene: Retrieves all the gene objects for the regions. If query
param histogram=true, frequency values per genomic interval will be
returned instead.
    - get_model: Get JSON specification of Variant data model
    - get_regulatory: Retrieves all regulatory elements in a region
    - get_repeat: Retrieves all repeats for the regions
    - get_sequence: Retrieves genomic sequence
    - get_tfbs: Retrieves all transcription factor binding site objects
for the regions. If query param histogram=true, frequency values per
genomic interval will be returned instead.
    - get_transcript: Retrieves all transcript objects for the regions
    - get_variation: Retrieves all the variant objects for the regions. If
query param histogram=true, frequency values per genomic interval will be
returned instead.
```

We can get the accepted parameters and filters for a specific method of interest by using the *get_help* me thod:

```
>>> cbc.get_region_client().get_help('get_gene', show_params=True)
```

PyCellBase API

PyCellBase implements a *CellBaseClient* object that acts as a factory to create all different clients such as RegionClient or GeneClient. Each of this clients implements a Python function to query each REST web service, you can see all web services at http://bioinfo.hpc.cam.ac.uk/cellbase/webservices/